

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

- STD. FILE 100

(2)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER	12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: DDC, Inc. DACS-80186, Version 4.3, MicroVAX II (host) to Intel 80186 iSBC 186/03A (target), 890324S1.10067		5. TYPE OF REPORT & PERIOD COVERED 24 March 1989 - 1 Dec 1990
7. AUTHOR(s) National Bureau of Standards, Gaithersburg, Maryland, USA		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards, Gaithersburg, Maryland, USA		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) ational Bureau of Standards, aithersburg, Maryland, USA		12. REPORT TYPE
		13. NUMBER OF PAGES
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

1. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

2. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)

UNCLASSIFIED

3. SUPPLEMENTARY NOTES

DTIC
ELECTE
MAY 30 1989
S E D

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada
Compiler Validation Capability, ACVC, Validation Testing, Ada
Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-
1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DACS-80186, Version 4.3, DDC, Inc., National Institute of Standards and Technology,
MicroVAX II under MicroVMS, Version 4.6 (host) to Intel 80186 iSBC 186/03A under
Bare (target), ACVC 1.10

89 5 30 006

DD FORM 1473
1 JAN 73EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-8601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A208 514

AVF Control Number: NIST89VDDC530_1.10
DRAFT COMPLETED: 03-31-89
FINAL COMPLETED: 04-25-89

Ada Compiler Validation Summary Report:

Compiler Name: DACS-80186, Version 4.3

Certificate Number: 890324S1.10067

Host: MicroVAX II under MicroVMS, Version 4.6

Target: Intel 80186 iSBC 186/03A under Bare

Testing Completed 24 March 1989 Using ACVC 1.10

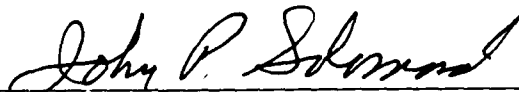
This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems Laboratory (NCSL)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond
Director
Washington D.C. 20301

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

AVF Control Number: NIST89VDDC530_1.10
DRAFT COMPLETED: 03-31-89
FINAL COMPLETED: 04-25-89

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890324S1.10067
DDC, Inc.
DACS-80186, Version 4.3
MicroVAX II Host and Intel 80186 iSBC 186/03A Target

Completion of On-Site Testing:
24 March 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

AVF Control Number: NIST89VDDC530_1.10
DRAFT COMPLETED: 03-31-89
FINAL COMPLETED: 04-25-89

Ada Compiler Validation Summary Report:

Compiler Name: DACS-80186, Version 4.3

Certificate Number: 890324SI.10067

Host: MicroVAX II under MicroVMS, Version 4.6

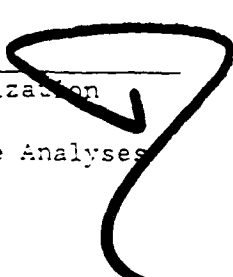
Target: Intel 80186 iSBC 186/03A under Bare

Testing Completed 24 March 1989 Using ACVC 1.10

This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems Laboratory (NCSL)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Washington D.C. 20301

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	IMPLEMENTATION CHARACTERISTICS	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS . .	3-6
3.7	ADDITIONAL TESTING INFORMATION	3-7
3.7.1	Prevalidation	3-7
3.7.2	Test Method	3-7
3.7.3	Test Site	3-8
APPENDIX A	CONFORMANCE STATEMENT	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	
APPENDIX E	COMPILER OPTIONS AS SUPPLIED BY DDC-I, Inc.	

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 24 March 1989 at Phoenix, Arizona.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure

consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved

words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity

functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated.

A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2
CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: DACS-80186, Version 4.3

ACVC Version: 1.10

Certificate Number: 890324S1.10067

Host Computer:

Machine: MicroVAX II

Operating System: MicroVMS, Version 4.6

Memory Size: 16MBytes

Target Computer:

Machine:

Board: Intel 80186 iSBC 186/03A

CPU: 80186

Bus: MULTIBUS 1

I/O: 8274

Timer: 80130

Operating System: Bare

Memory Size: 1MByte

Communications Network: The host computer, a MicroVAX II, was linked via ETHERNET to an IBM PC XT which is connected to the target computer, an Intel 80186 iSBC 186/03A, via an in-circuit emulator (I2ICE).

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

- (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)
- (3) The compiler rejects tests containing block statements nested to 65 levels. (See test D56001B.)
- (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

- (1) This implementation supports the additional predefined types `SHORT_INTEGER`, `LONG_FLOAT`, and `LONG_INTEGER` in the package `STANDARD`. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- (1) All of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- (2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

- (3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)
- (4) `NUMERIC_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- (5) `NUMERIC_ERROR` is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)
- (6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- (1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)
- (2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)
- (3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

- (1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises `NUMERIC_ERROR`. (See test C36003A.)
- (2) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components. (See test C36202A.)
- (3) `NUMERIC_ERROR` is raised when `'LENGTH` is applied to an array type with `SYSTEM.MAX_INT + 2` components. (See test C36202B.)

- (4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when declaring two packed Boolean arrays with INTEGER'LAST + 3 components. (See test C52103X.)
- (5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)
- (6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)
- (7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)
- (8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

- (1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

- (1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)
- (2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- (3) CONSTRAINT_ERROR is raised before all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragas.

- (1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

- (1) Generic specifications and bodies cannot be compiled in separate compilations. (See tests CA2009C, CA2009F, BC3204C, and BC3205D.)

Generic package declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

- (2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)
- (3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA1012A.)
- (4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- (5) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)
- (6) Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)
- (7) Generic library package specifications and bodies cannot be compiled in separate compilations. (See tests BC3204C and BC3205D.)
- (8) Generic non-library package bodies as subunits cannot be compiled in separate compilations. (See test CA2009C.)
- (9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

- (1) The package SEQUENTIAL_IO cannot be instantiated with

unconstrained array types and record types with discriminants without defaults. (See tests EE2201D and EE2201E.)

- (2) The package `DIRECT_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests EE2401D and EE2401G.)
- (3) `USE_ERROR` is raised when Mode `IN_FILE` is not supported for the operation of `CREATE` for `SEQUENTIAL_IO`. (See test CE2102D.)
- (4) `USE_ERROR` is raised when Mode `IN_FILE` is not supported for the operation of `CREATE` for `DIRECT_IO`. (See test CE2102I.)
- (5) `USE_ERROR` is raised when Mode `IN_FILE` is not supported for the operation of `CREATE` for text files. (See test CE3102E.)
- (6) Modes `IN_FILE` and `OUT_FILE` not are supported for text files. (See tests CE3102E and CE3102I.)
- (7) `RESET` and `DELETE` operations are not supported for `SEQUENTIAL_IO`. (See tests CE2102G and CE2102X.)
- (8) `RESET` and `DELETE` operations are not supported for `DIRECT_IO`. (See tests CE2102K and CE2102Y.)
- (9) `RESET` and `DELETE` operations are not supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)
- (10) Only one internal file can be associated with each external file for sequential files when writing only. (See tests CE2102L.)
- (11) More than one internal file can be associated with the each external file for direct files when writing or reading. (See tests CE2107F.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 43 tests had been withdrawn because of test errors. The AVF determined that 657 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 245 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 73 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	123	1132	1685	16	15	46	3017
Inapplicable	6	6	631	1	13	0	657
Withdrawn	1	2	34	0	6	0	43
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER														TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14		
Passed	195	572	554	247	172	99	161	332	135	36	250	188	76	3017	
Inapplicable	17	77	126	1	0	0	5	1	2	0	2	181	245	657	
Wdrn	1	1	0	0	0	0	0	1	0	0	1	35	4	43	
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717	

3.4 WITHDRAWN TESTS

The following 43 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

```

A39005G  B97102E  BC3009B  CD2A62D  CD2A63A  CD2A63B
CD2A63C  CD2A63D  CD2A66A  CD2A66B  CD2A66C  CD2A66D
CD2A73A  CD2A73B  CD2A73C  CD2A73D  CD2A76A  CD2A76B
CD2A76C  CD2A76D  CD2A81G  CD2A83G  CD2A84M  CD2A84N
CD2B15C  CD2D11B  CD5007B  CD50110  CD7105A  CD7203B
CD7204B  CD7205C  CD7205D  CE2107I  CE3111C  CE3301A
CE3411B  E28005C  ED7004B  ED7005C  ED7005D  ED7006C
ED7006D

```

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 657 tests were inapplicable for the reasons indicated:

The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

```

C24113L..Y (14 tests)    C35705L..Y (14 tests)
C35706L..Y (14 tests)    C35707L..Y (14 tests)

```

C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

C24113I..K (3 tests) are not applicable because the line length of the input file must not exceed 126 characters.

A39005E, C87B62C, CD1009L, CD1C03F, CD2D11A, and CD2D13A (6 tests) are not applicable because 'SMALL clause is not supported.

C35508I, C35508J, C35508M, C35508N, AD1C04D, AD3015C, AD3015F, AD3015H, AD3015K, CD1C04B, CD1C04C, CD1C04E, CD2A23C, CD2A23D, CD2A24C, CD2A24D, CD2A24G, CD2A24H, CD3015A, CD3015B, CD3015D, CD3015E, CD3015G, CD3015I, CD3015J, CD3015L, CD4051A, CD4051B, CD4051C, and CD4051D (30 tests) are not applicable because this implementation does not support the specified change in representation for derived types.

C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

B86001X, C45231D, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

D56001B uses 65 levels of block nesting which exceeds the capacity of the compiler.

B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

C45531M, C45531N, C45532M, and C45532N use fine 48 bit fixed point base types which are not supported by this compiler.

C45531O, C45531P, C45532O, and C45532P use coarse 48 bit fixed point base types which are not supported by this compiler.

C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

CA2009C is not applicable because this implementation does not

permit compilation of generic non-library package bodies in separate files from their specifications.

CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies in separate files from their specifications.

BC3204C and BC3205D are not applicable because this implementation does not permit compilation of generic library package bodies in separate files from their specifications.

CD1009C, CD2A41A..B, CD2A41E, and CD2A42A..J (14 tests) are not applicable because this implementation does not support the 'SIZE clause for floating-point types.

CD2A51A..B, CD2A51D..E, CD2A52A..D, CD2A52G..J, CD2A53A..E, CD2A54A..D, CD2A54G..J, and ED2A56A (26 tests) are not applicable because this implementation does not support the 'SIZE clause for a fixed-point types.

CD2A61A..L, CD2A62A..C, CD2A64A..D, CD2A65A..D, CD2A71A..D, CD2A72A..D, CD2A74A..D, and CD2A75A..D (39 tests) are not applicable because this implementation does not support the 'SIZE clause for an array type which does not imply compression of inter-component gaps.

CD2A84B..I and CD2A84K..L (10 tests) are not applicable because this implementation does not support the 'SIZE clause for an access type.

CD5003B..I, CD5011A, CD5011C, CD5011E, CD5011G, CD5011I, CD5011K, CD5011M, CD5011Q, CD5012A..B, CD5012E..F, CD5012I, CD5012M, CD5013A, CD5013C, CD5013E, CD5013G, CD5013I, CD5013K, CD5013M, CD5013O, CD5014T, and CD5014V..Z (36 tests) are not applicable because this implementation does not support non-static address clauses for a variable.

CD5011B, CD5011D, CD5011F, CD5011H, CD5011L, CD5011N, CD5011R, CD5011S, CD5012C..D, CD5012G..H, CD5012L, CD5013B, CD5013D, CD5013F, CD5013H, CD5013L, CD5013N, CD5013R, and CD5014U (21 tests) are not applicable because this implementation does not support non-static address clauses for a constant.

CD5012J, CD5013S, and CD5014S (3 tests) are not applicable because this implementation does not support non-static address clauses.

CD4041A is not applicable because this implementation does not support the alignment clauses for alignments other than SYSTEM.STORAGE_UNIT for record representation clauses.

The following 242 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102A..C (3 tests)	CE2102G..H (2 tests)
CE2102K	CE2102N..Y (12 tests)
CE2103C..D (2 tests)	CE2104A..D (4 tests)
CE2105A..B (2 tests)	CE2106A..B (2 tests)
CE2107A..H (8 tests)	CE2107L
CE2108A..H (8 tests)	CE2109A..C (3 tests)
CE2110A..D (4 tests)	CE2111A..I (9 tests)
CE2115A..B (2 tests)	CE2201A..C (3 tests)
EE2201D..E (2 tests)	CE2201F..N (9 tests)
CE2204A..D (4 tests)	CE2205A
CE2208B	CE2401A..C (3 tests)
EE2401D	CE2401E..F (2 tests)
EE2401G	CE2401H..L (5 tests)
CE2404A..B (2 tests)	CE2405B
CE2406A	CE2407A..B (2 tests)
CE2408A..B (2 tests)	CE2409A..B (2 tests)
CE2410A..B (2 tests)	CE2411A
CE3102A..B (2 tests)	EE3102C
CE3102F..H (3 tests)	CE3102J..K (2 tests)
CE3103A	CE3104A..C (3 tests)
CE3107B	CE3108A..B (2 tests)
CE3109A	CE3110A
CE3111A..B (2 tests)	CE3111D..E (2 tests)
CE3112A..D (4 tests)	CE3114A..B (2 tests)
CE3115A	EE3203A
CE3208A	EE3301B
CE3302A	CE3305A
CE3402A	EE3402B
CE3402C..D (2 tests)	CE3403A..C (3 tests)
CE3403E..F (2 tests)	CE3404B..D (3 tests)
CE3405A	EE3405B
CE3405C..D (2 tests)	CE3406A..D (4 tests)
CE3407A..C (3 tests)	CE3408A..C (3 tests)
CE3409A	CE3409C..E (3 tests)
EE3409F	CE3410A
CE3410C..E (3 tests)	EE3410F
CE3411A	CE3411C
CE3412A	EE3412C
CE3413A	CE3413C
CE3602A..D (4 tests)	CE3603A
CE3604A..B (2 tests)	CE3605A..E (5 tests)
CE3606A..B (2 tests)	CE3704A..F (6 tests)
CE3704M..O (3 tests)	CE3706D
CE3706F..G (2 tests)	CE3804A..P (16 tests)
CE3805A..B (2 tests)	CE3806A..B (2 tests)
CE3806D..E (2 tests)	CE3806G..H (2 tests)
CE3905A..C (3 tests)	CE3905L
CE3906A..C (3 tests)	CE3906E..F (2 tests)

CE2103A, CE2103B, and CE3107A (3 tests) are not applicable because these tests expect that CREATE ..., ..., <bad_file_name> will cause

only NAME_ERROR to be raised; but for implementations that do not support file I/O, it is preferable that USE_ERROR be raised, and that is what this implementation does.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 73 tests.

The following 64 tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B26001A	B26002A	B26005A	B28001D	B29001A
B2A003B	B2A003C	B33301B	B35101A	B37106A	B37301B
B37302A	B38003A	B38003B	B38009A	B38009B	B51001A
B53009A	B54A01C	B54A01H	B55A01A	B61001C	B61001D
B61001F	B61001H	B61001I	B61001M	B61001R	B61001S
B61001W	B67001H	B91001A	B91002A	B91002B	B91002C
B91002D	B91002E	B91002F	B91002G	B91002H	B91002I
B91002J	B91002K	B91002L	B95030A	B95061A	B95061F
B95061G	B95077A	B97103E	B97104G	BA1101B	BC1109A
BC1109C	BC1109D	BC1202A	BC1202B	BC1202E	BC1202F
BC1202G	BC2001D	BC2001E	BC3009A		

The following 9 tests contain modifications to their respective source code files:

AD7006A wrongly assumes that an expression in an assignment statement is of type universal integer, and so should deliver a correct result that is in the range of type INTEGER. This implementation is correct in treating the expression as being of type INTEGER; an exception is raised because the operand SYSTEM.MEMORY_SIZE exceeds INTEGER'LAST.

The implementer's modification of this test (declaring the assigned -- variable I to be of type LONG_INTEGER) is ruled to be an acceptable means to passing this test by the AJPO.

C34007A, C34007D, C34007G, C34007J, C34007M, C34007P, C34007S, and C87B62B (8 tests) The AVO accepts the implementer's argument that, without there being a STORAGE_SIZE length clause for an access type, the meaning of the attribute 'STORAGE_SIZE is undefined for

that a type.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the DACS-80186, Version 4.3 compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the DACS-80186, Version 4.3 compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	MicroVAX II
Host operating system:	MicroVMS, Version 4.6
Target computer:	Intel 80186 iSBC 186/03A
Target operating system:	Bare
Compiler:	DACS-80186, Version 4.3
Pre-linker:	DACS-80186 LINKER
Assembler:	INTEL ASM86
Linker:	INTEL LINK86
Loader/Downloader:	INTEL LOC86
Runtime System:	DDC-I RTS

The host computer, a MicroVAX II, was linked via ETHERNET to an IBM PC XT which is connected to the target computer, an Intel 80186 iSBC 186/03A, via an in-circuit emulator (I2ICE).

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

TEST INFORMATION

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the MicroVAX II, and all executable tests were run on the Intel 80186 iSBC 186/03A. Results were printed from the host computer.

The compiler was tested using command scripts provided by DDC, Inc. and reviewed by the validation team. The compiler was tested using the following option settings. See Appendix E for a complete listing of the compiler options for this implementation.

/LIST
/NOSAVE

Tests were compiled, linked, and executed (as appropriate) using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. Selected listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Phoenix, Arizona and was completed on 24 March 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

DDC, Inc. has submitted the following Declaration of Conformance concerning the DACS-80186, Version 4.3.

DECLARATION OF CONFORMANCE

Compiler Implementer: DDC-I, Inc.
Ada Validation Facility: National Institute of Standards and
Technology
ACVC Version: ACVC 1.10

Base Configuration

Base Compiler Name: DACS-80186 Version: 4.3
Host Architecture - ISA: MicroVAX II OS&VER: MicroVMS 4.6
Target Architecture - ISA: Intel 80186 iSBC 186/03A OS&VER: Bare

Derived Compiler Registration

For the following derived compilers, both the input files for the ACVC tests and the result files for the ACVC tests are the same as that for the base configuration.

Base Compiler Name: DACS-80186 Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0,
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel 80186 iSBC 186/03A OS&VER: Bare

Base Compiler Name: DACS-8086 Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0,
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel 8086 iSBC 86/05A OS&VER: Bare

Base Compiler Name: DACS-80286 Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0,
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel 80286 iSBC 286/12 OS&VER: Bare

Base Compiler Name: DACS-80286 Protected Mode
Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0,
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel 80286 iSBC 286/12
OS&VER: Bare (Protected Mode)

Base Compiler Name: DACS-80386 Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel Multibus II 80386 iSBC 386/116
OS&VER: Bare

Base Compiler Name: DACS-80386 Version: 4.3
Host Architecture - ISA: Complete DEC Family of Vax,
Vax Station, and MicroVax Computers
OS&VER: Vax/VMS 4.6 & 5.0
or MicroVMS 4.6 & 5.0
Target Architecture - ISA: Intel Multibus I 80386 iSBC 386/21
OS&VER: Bare

Implementer's Declaration

I, the undersigned, representing DDC-I, Inc. have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare the DDC-I, Inc. is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Lee Silverthorn
Lee Silverthorn, President - DDC-I, Inc.

3/24/89
Date

Owner's Declaration

I, the undersigned, representing DDC-I, Inc. take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler(s) and concur with the contents.

Lee Silverthorn
Lee Silverthorn, President - DDC-I, Inc.

3/24/89
Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DACS-80186, Version 4.3 compiler, as described in this Appendix, are provided by DDC-I, Inc.. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

...

```
type INTEGER is range -32_768 .. 32_767 ;
type SHORT_INTEGER is range -128 .. 127;
type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_647;

type FLOAT is digits 6 range
    -3.40282366920938E+38 .. 3.40282366920938E+38;
type LONG_FLOAT is digits 15 range
    -1.7976931348623157E+308 .. 1.7976931348623157E+308;

type DURATION is delta 6.10351562500000E-05 range
    -131_072.00000 .. 131_071.00000 ;
```

...

end STANDARD;

APPENDIX F IMPLEMENTATION-DEPENDENT CHARACTERISTICS

This appendix describes the implementation-dependent characteristics of DACS-80X86[®] as required in Appendix F of the Ada Reference Manual (ANSI/MIL-STD-1815A).

F.1 Implementation-Dependent Pragmas

This section describes all implementation defined pragmas.

F.1.1 Pragma INTERFACE_SPELLING

This pragma allows an Ada program to call a non-Ada program whose name contains characters that would be an invalid Ada subprogram identifier. This pragma must be used in conjunction with pragma INTERFACE, i.e., pragma INTERFACE must be specified for the non-Ada subprogram name prior to using pragma INTERFACE_SPELLING.

The pragma has the format:

```
pragma INTERFACE_SPELLING (subprogram name,  
                           string literal);
```

where the subprogram name is that of one previously given in pragma INTERFACE and the string literal is the exact spelling of the interfaced subprogram in its native language. This pragma is only required when the subprogram name contains invalid characters for Ada identifiers.

Example:

```
function RTS_GetDataSegment return Integer;  
  
pragma INTERFACE      (ASM86, RTS_GetDataSegment);  
pragma INTERFACE_SPELLING (RTS_GetDataSegment,  
                           "R1SMGS?GetDataSegment");
```

F.1.2 Pragma INTERRUPT_HANDLER

This pragma will cause the compiler to generate fast interrupt handler entries instead of the normal task calls for the entries in the task in which it is specified. It has the format:

```
pragma INTERRUPT_HANDLER;
```

User's Guide
Appendix F

The pragma must appear as the first thing in the specification of the task object. The task must be specified in a package and not a procedure. See section F.6.2 for more details and restrictions on specifying address clauses for task entries.

F.1.3 Pragma LT_STACK_SPACE

This pragma sets the size of a library task stack segment. The pragma has the format:

```
pragma LT_STACK_SPACE (T, N);
```

where T denotes either a task object or task type and N designates the size of the library task stack segment in words.

The library task's stack segment defaults to the size of the library task stack. The size of the library task stack is normally specified via the representation clause

```
for T'SORAGE_SIZE use N;
```

The size of the library task stack segment determines how many tasks can be created which are nested within the library task. All tasks created within a library task will have their stacks allocated from the same segment as the library task stack. Thus, pragma LT_STACK_SPACE must be specified to reserve space within the library task stack segment so that nested tasks' stacks may be allocated.

The following restrictions are places on the use of LT_STACK_SPACE:

- 1) It must be used only for library tasks.
- 2) It must be placed immediately after the task object or type name declaration.
- 3) The library task stack segment size (N) must be greater than or equal to the library task stack size.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined.

F.3 Package SYSTEM

The specification of the package SYSTEM for the 80x86 Real Address Mode and 80286 Protected Mode systems is:

User's Guide
Appendix F

package System is

```
type Word is new Integer;
type LongWord is new Long_Integer;
```

```
type UnsignedWord is range 0..65535;
for UnsignedWord'SIZE use 16;
```

```
subtype SegmentId is UnsignedWord;
```

```
type Address is record
  offset : UnsignedWord;
  segment : SegmentId;
end record;
```

```
subtype Priority is Word range 0..31;
```

```
type Name is (iAPX86,
               iAPX186,
               iAPX286,
               iAPX386);
```

```
System_Name : constant Name := iAPX186;
Storage_Unit : constant := 16;
Memory_Size : constant := 1_048_576;
Min_Int : constant := -2_147_483_647-1;
Max_Int : constant := 2_147_483_647;
Max_Digits : constant := 15;
Max_Mantissa : constant := 31;
Fine_Delta : constant := 2.0 / MAX_INT;
Tick : constant := 0.000_000_125;
```

```
type Interface_Language is (PLM86, ASM86);
```

```
type ExceptionId is record
  unit_number : UnsignedWord;
  unique_number : UnsignedWord;
end record;
```

```
type TaskValue is new Integer;
type AccTaskValue is access TaskValue;
```

```
type Semaphore is
  record
    counter : UnsignedWord;
    first : TaskValue;
    last : TaskValue;
  end record;
```

```
InitSemaphore : constant Semaphore'(1, 0, 0);
```

```
end SYSTEM;
```

User's Guide
Appendix F

The package SYSTEM specification for the 80386 Protected Mode system is:

package System is

```

type      Word          is new Short_Integer;
type      DWord         is new Integer;
type      QWord         is new Long_Integer

type      UnsignedWord  is range 0..65535;
for       UnsignedWord'SIZE use 16;
type      UnsignedDword is range 0..16#7FFF_FFFF#;
for       UnsignedDword'SIZE use 32;

subtype SegmentId      is UnsignedWord;

type      Address is record
    offset   : UnsignedDword;
    segment  : SegmentId;
end record;

subtype Priority      is Word range 0..31;

type      Name        is (iAPX86,
                          iAPX186,
                          iAPX286,
                          iAPX386,
                          iAPX386_SM,
                          iAPX386_FM);

System_Name      :      constant Name      := iAPX386_SM;
Storage_Unit     :      constant           := 16;
Memory_Size      :      constant           := 1_048_576;
Min_Int          :      constant := -16#8000_0000_0000_0000#;
Max_Int          :      constant := 16#7FFF_FFFF_FFFF_FFFF#;
Max_Digits       :      constant           := 15;
Max_Mantissa     :      constant           := 31;
Fine_Delta       :      constant           := 2#1.0#E-31;
Tick             :      constant           := 0.000_000_125;

type      Interface_Language is (PLM86, ASM86,
                                C86, C86_REVERSE);

type      ExceptionId is record
    unit_number   : UnsignedDword;
    unique_number : UnsignedDword;
end record;

type      TaskValue      is new Integer;
type      AccTaskValue   is access TaskValue;

```

User's Guide
Appendix F

```
type      Semaphore      is
  record
    counter      :  UnsignedWord;
    first        :  TaskValue;
    last         :  TaskValue;
  end record;

InitSemaphore : constant Semaphore'(1, 0, 0);

end SYSTEM;
```

F.4 Representation Clauses

The DACS-80x86® fully supports the 'SIZE representation for derived types. The representation clauses that are accepted for non-derived types are described in the following subsections.

F.4.1 Length Clause

Some remarks on implementation dependent behavior of length clauses are necessary:

- When using the SIZE attribute for discrete types, the maximum value that can be specified is 16 bits.
- SIZE is only obeyed for discrete types when the type is a part of a composite object, e.g. arrays or records, for example:

```
type byte is range 0..255;
for byte'size use 8;

sixteen_bits_allocated : byte;           -- one word
                                         -- allocated

eight_bit_per_element : array(0..7) of byte;
                                         -- four words
                                         -- allocated

type rec is
  record
    c1,c2 : byte;                        -- eight bits per
                                         -- component
  end record;
```

User's Guide
Appendix F

- Using the STORAGE_SIZE attribute for a collection will set an upper limit on the total size of objects allocated in this collection. If further allocation is attempted, the exception STORAGE_ERROR is raised.
- When STORAGE_SIZE is specified in a length clause for a task, the process stack area will be of the specified size. The process stack area will be allocated inside the "standard" stack segment.

F.4.2 Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of INTEGER'FIRST + 1..INTEGER'LAST - 1.

F.4.3 Record Representation Clauses

When representation clauses are applied to records the following restrictions are imposed:

- the component type is a discrete type different from LONG_INTEGER
- the component type is an array with a discrete element type different from LONG_INTEGER
- the storage unit is 16 bits
- a record occupies an integral number of storage units
- a record may take up a maximum of 32K storage units
- a component must be specified with its proper size (in bits), regardless of whether the component is an array or not.
- if a non-array component has a size which equals or exceeds one storage unit (16 bits) the component must start on a storage unit boundary, i.e. the component must be specified as:

component at N range 0..16 * M - 1;

where N specifies the relative storage unit number (0,1,...) from the beginning of the record, and M the required number of storage units (1,2,...)

- the elements in an array component should always be wholly contained in one storage unit

User's Guide
Appendix F

- if a component has a size which is less than one storage unit, it must be wholly contained within a single storage unit:

component at N range X .. Y;

where N is as in previous paragraph, and $0 \leq X \leq Y \leq 15$

When dealing with PACKED ARRAY the following should be noted:

- the elements of the array are packed into 1,2,4 or 8 bits

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the component with the value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

F.4.3.1 Alignment Clauses

Alignment clauses for records are implemented with the following characteristics:

- If the declaration of the record type is done at the outermost level in a library package, any alignment is accepted.
- If the record declaration is done at a given static level (higher than the outermost library level, i.e., the permanent area), only word alignments are accepted.
- Any record object declared at the outermost level in a library package will be aligned according to the alignment clause specified for the type. Record objects declared elsewhere can only be aligned on a word boundary. If the record type has been associated a different alignment, an error message will be issued.
- If a record type with an associated alignment clause is used in a composite type, the alignment is required to be one word; an error message is issued if this is not the case.

F.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

F.6 Address Clauses

This section describes the implementation of address clauses and what types of entities may have their address specified by the user.

F.6.1 Objects

Address clauses are supported for scalar and composite objects whose size can be determined at compile time.

F.6.2 Task Entries

The implementation supports two methods to equate a task entry to a hardware interrupt through an address clause:

- 1) Direct transfer of control to a task accept statement when an interrupt occurs (requires use of the pragma `INTERRUPT_HANDLER`).
- 2) Mapping of an interrupt onto a normal conditional entry call, i.e., the entry can be called from other tasks without special actions, as well as being called when an interrupt occurs.

F.6.2.1 Fast Interrupt Entry

Directly transferring control to an accept statement when an interrupt occurs requires the implementation dependent pragma `INTERRUPT_HANDLER` to tell the compiler that the task is an interrupt handler. By using this pragma, the user is agreeing to place certain restrictions on the task in order to speed up the software response to the hardware interrupt. Consequently, use of this method to capture interrupts is much more efficient than the general method.

The following constraints are placed on the task:

- 1) It must be a task object, i.e., not a task type.
- 2) The pragma must appear first in the specification of the task object.
- 3) All entries of the task object must be single entries with no parameters.
- 4) The entries must not be called from any task.

User's Guide
Appendix F

- 5) The body of the task object must not contain anything other than simple accept statements (potentially enclosed in a loop) referencing only global variables, i.e., no local variables. In the statement list of a simple accept statement, it is allowed to call normal, single and parameterless, entries of other tasks, but no other tasking constructs are allowed. The call to another task entry, in this case, will not lead to an immediate task context switch, but will return to the caller when complete. Once the accept is completed, the task priority rules will be obeyed, and a context switch may occur.

F.6.2.2 Normal Interrupt Entry

Mapping of an interrupt onto a normal conditional entry call puts the following constraints on the involved entries and tasks:

- 1) The affected entries must be defined in a task object only (not a task type).
- 2) The entries must be single and parameterless.

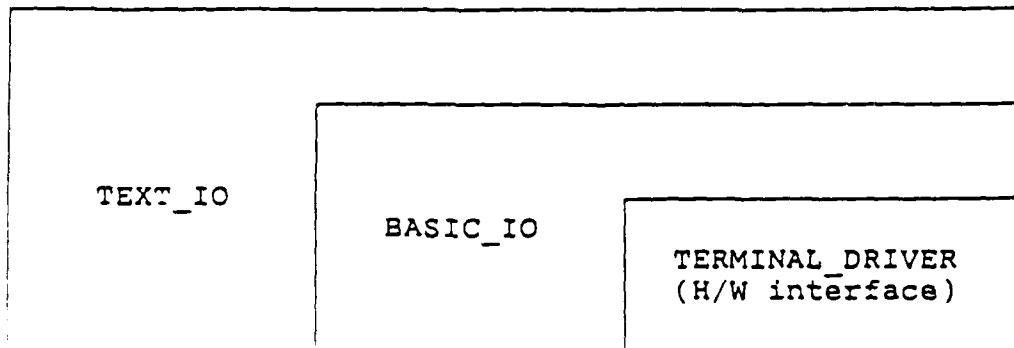
Any interrupt entry, which is not found in an interrupt handler (first method), will lead to an update of the interrupt vector segment at link time. The interrupt vector segment will be updated to point to the interrupt routine generated by the compiler to make the task entry call. The interrupt vector segment is part of the user configurable data and consists of a segment, preset to the "standard" interrupt routines (e.g., `constraint_error`). See section 7.2.15 (RTS Configuration of Interrupt Vector Ranges) for details on how to specify interrupt vector ranges.

F.7 Unchecked Conversions

Unchecked conversion is only allowed between objects of the same "size".

F.8 Input/Output Packages

In many embedded systems, there is no need for a traditional I/O system, but in order to support testing and validation, DDC-I has developed a small terminal oriented I/O system. This I/O system consists essentially of TEXT_IO, adapted with respect to handling only a terminal and not file I/O (file I/O will cause a USE error to be raised), and a low level package called TERMINAL_DRIVER. A BASIC_IO package has been provided for convenience purposes, forming an interface between TEXT_IO and TERMINAL_DRIVER as illustrated in the following figure.



The TERMINAL_DRIVER package is the only package that is target dependent, i.e., it is the only package that need be changed when changing communications controllers. The actual body of the TERMINAL_DRIVER is written in assembly language, but an Ada interface to this body is provided. A user can also call the terminal driver routines directly, i.e., from an assembly language routine. TEXT_IO and BASIC_IO are written completely in Ada and need not be changed.

BASIC_IO provides a mapping between TEXT_IO control characters and ASCII as follows:

TEXT_IO	ASCII Character
LINE_TERMINATOR	ASCII.CR
PAGE_TERMINATOR	ASCII.FF
FILE_TERMINATOR	ASCII.EM (ctrl Z)
NEW_LINE	ASCII.LF

User's Guide
Appendix F

The services provided by the terminal driver are:

- 1) Reading a character from the communications port.
- 2) Writing a character to the communications port.

The terminal driver comes in two versions: one which supports tasking, i.e., asynchronous I/O, and a version which assumes no tasking.

F.8.1 Package TEXT_IO

The specification of package TEXT_IO:

```
pragma page;
with BASIC_IO;

with IO_EXCEPTIONS;
package TEXT_IO is

    type FILE_TYPE is limited private;

    type FILE_MODE is (IN_FILE, OUT_FILE);

    type COUNT is range 0 .. INTEGER'LAST;
    subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;
    UNBOUNDED: constant COUNT:= 0; -- line and page length

    -- max. size of an integer output field 2#....#
    subtype FIELD is INTEGER range 0 .. 35;

    subtype NUMBER_BASE is INTEGER range 2 .. 16;

    type TYPE_SET is (LOWER_CASE, UPPER_CASE);

pragma PAGE;
-- File Management

    procedure CREATE (FILE : in out FILE_TYPE;
                     MODE : in FILE_MODE := OUT_FILE;
                     NAME : in STRING := "";
                     FORM : in STRING := "")
    );

    procedure OPEN (FILE : in out FILE_TYPE;
                   MODE : in FILE_MODE;
                   NAME : in STRING;
                   FORM : in STRING := "")
    );
```

User's Guide
Appendix F

```
procedure CLOSE (FILE : in out FILE_TYPE;)  
procedure DELETE (FILE : in out FILE_TYPE);  
procedure RESET (FILE : in out FILE_TYPE;  
                 MODE : in FILE_MODE);  
procedure RESET (FILE : in out FILE_TYPE);  
  
function MODE (FILE : in FILE_TYPE) return FILE_MODE;  
function NAME (FILE : in FILE_TYPE) return STRING;  
function FORM (FILE : in FILE_TYPE) return STRING;  
  
function IS_OPEN(FILE : in FILE_TYPE return BOOLEAN;  
  
pragma PAGE;  
-- control of default input and output files  
  
procedure SET_INPUT (FILE : in FILE_TYPE);  
procedure SET_OUTPUT (FILE : in FILE_TYPE);  
  
function STANDARD_INPUT return FILE_TYPE;  
function STANDARD_OUTPUT return FILE_TYPE;  
  
function CURRENT_INPUT return FILE_TYPE;  
function CURRENT_OUTPUT return FILE_TYPE;  
  
pragma PAGE;  
-- specification of line and page lengths  
  
procedure SET_LINE_LENGTH (FILE : in FILE_TYPE;  
                           TO : in COUNT);  
procedure SET_LINE_LENGTH (TO : in COUNT);  
  
procedure SET_PAGE_LENGTH (FILE : in FILE_TYPE;  
                           TO : in COUNT);  
procedure SET_PAGE_LENGTH (TO : in COUNT);  
  
function LINE_LENGTH (FILE : in FILE_TYPE)  
    return COUNT;  
function LINE_LENGTH return COUNT;  
  
function PAGE_LENGTH (FILE : in FILE_TYPE)  
    return COUNT;  
function PAGE_LENGTH return COUNT;  
  
pragma PAGE;  
-- Column, Line, and Page Control  
  
procedure NEW_LINE (FILE : in FILE_TYPE;  
                   SPACING : in POSITIVE_COUNT := 1);  
procedure NEW_LINE (SPACING : in POSITIVE_COUNT := 1);  
  
procedure SKIP_LINE (FILE : in FILE_TYPE;  
                   SPACING : in POSITIVE_COUNT := 1);
```

User's Guide
Appendix F

```
procedure SKIP_LINE (SPACING : in POSITIVE_COUNT := 1);

function END_OF_LINE (FILE : in FILE_TYPE) return BOOLEAN;
function END_OF_LINE                                     return BOOLEAN;

procedure NEW_PAGE   (FILE : in FILE_TYPE);
procedure NEW_PAGE   ;

procedure SKIP_PAGE   (FILE : in FILE_TYPE);
procedure SKIP_PAGE   ;

function END_OF_PAGE (FILE : in FILE_TYPE) return BOOLEAN;
function END_OF_PAGE                                     return BOOLEAN;

function END_OF_FILE (FILE : in FILE_TYPE) return BOOLEAN;
function END_OF_FILE                                     return BOOLEAN;

procedure SET_COL      (FILE : in FILE_TYPE;
                        TO   : in POSITIVE_COUNT);
procedure SET_COL      (TO   : in POSITIVE_COUNT);

procedure SET_LINE     (FILE : in FILE_TYPE;
                        TO   : in POSITIVE_COUNT);
procedure SET_LINE     (TO   : in POSITIVE_COUNT);

function COL           (FILE : in FILE_TYPE)
                        return POSITIVE_COUNT;
function COL           return POSITIVE_COUNT;

function LINE          (FILE : in FILE_TYPE)
                        return POSITIVE_COUNT;
function LINE          return POSITIVE_COUNT;

function PAGE          (FILE : in FILE_TYPE)
                        return POSITIVE_COUNT;
function PAGE          return POSITIVE_COUNT;

pragma PAGE;
-- Character Input-Output

procedure GET  (FILE : in FILE_TYPE; ITEM : out CHARACTER);
procedure GET  (      ITEM : out CHARACTER);
procedure PUT  (FILE : in FILE_TYPE; ITEM : in CHARACTER);
procedure PUT  (      ITEM : in CHARACTER);

-- String Input-Output

procedure GET  (FILE : in FILE_TYPE; ITEM : out CHARACTER);
procedure GET  (      ITEM : out CHARACTER);
procedure PUT  (FILE : in FILE_TYPE; ITEM : in CHARACTER);
procedure PUT  (      ITEM : in CHARACTER);

procedure GET_LINE (FILE : in FILE_TYPE;
```

User's Guide
Appendix F

```
ITEM : out STRING;
LAST : out NATURAL));

procedure GET_LINE (ITEM : out STRING;
LAST : out NATURAL));

procedure PUT_LINE (FILE : in FILE_TYPE;
ITEM : in STRING);
procedure PUT_LINE (ITEM : in STRING);

pragma PAGE;
-- Generic Package for Input-Output of Integer Types

generic
type NUM is range <>;
package INTEGER_IO is

DEFAULT_WIDTH : FIELD := NUM'WIDTH;
DEFAULT_BASE : NUMBER_BASE := 10;

procedure GET (FILE : in FILE_TYPE;
ITEM : out NUM;
WIDTH : in FIELD := 0);
procedure GET (ITEM : out NUM;
WIDTH : in FIELD := 0);

procedure PUT (FILE : in FILE_TYPE;
ITEM : in NUM;
WIDTH : in FIELD := DEFAULT_WIDTH;
BASE : in NUMBER_BASE := DEFAULT_BASE);
procedure PUT (ITEM : in NUM;
WIDTH : in FIELD := DEFAULT_WIDTH;
BASE : in NUMBER_BASE := DEFAULT_BASE);

procedure GET (FROM : in STRING;
ITEM : out NUM;
LAST : out POSITIVE);

procedure PUT (TO : out STRING;
ITEM : in NUM;
BASE : in NUMBER_BASE := DEFAULT_BASE);

end INTEGER_IO;

pragma PAGE;
```

User's Guide
Appendix F

-- Generic Packages for Input-Output of Real Types

```
generic
  type NUM is digits <>;
package FLOAT_IO is

  DEFAULT_FORE : FIELD :=          2;
  DEFAULT_AFT  : FIELD := NUM'DIGITS - 1;
  DEFAULT_EXP  : FIELD :=          3;

  procedure GET  (FILE : in FILE_TYPE;
                  ITEM  : out NUM;
                  WIDTH : in FIELD := 0);
  procedure GET  (ITEM  : out NUM;
                  WIDTH : in FIELD := 0);

  procedure PUT  (FILE : in FILE_TYPE;
                  ITEM  : in NUM;
                  FORE  : in FIELD := DEFAULT_FORE;
                  AFT   : in FIELD := DEFAULT_AFT;
                  EXP   : in FIELD := DEFAULT_EXP);
  procedure PUT  (ITEM  : in NUM;
                  FORE  : in FIELD := DEFAULT_FORE;
                  AFT   : in FIELD := DEFAULT_AFT;
                  EXP   : in FIELD := DEFAULT_EXP);

  procedure GET  (FROM : in STRING;
                  ITEM  : out NUM;
                  LAST  : out POSITIVE);
  procedure PUT  (TO    : out STRING;
                  ITEM  : in NUM;
                  AFT   : in FIELD := DEFAULT_AFT;
                  EXP   : in FIELD := DEFAULT_EXP);

end FLOAT_IO;

pragma PAGE;
```

User's Guide
Appendix F

```
generic
  type NUM is delta <>;
package FIXED_IO is

  DEFAULT_FORE : FIELD := NUM'FORE;
  DEFAULT_AFT  : FIELD := NUM'AFT;
  DEFAULT_EXP  : FIELD := 0;

  procedure GET (FILE : in FILE_TYPE;
                 ITEM  : out NUM;
                 WIDTH : in FIELD := 0);
  procedure GET (ITEM : out NUM;
                 WIDTH : in FIELD := 0);

  procedure PUT (FILE : in FILE_TYPE;
                 ITEM  : in NUM;
                 FORE  : in FIELD := DEFAULT_FORE;
                 AFT   : in FIELD := DEFAULT_AFT;
                 EXP   : in FIELD := DEFAULT_EXP);

  procedure PUT (ITEM : in NUM;
                 FORE  : in FIELD := DEFAULT_FORE;
                 AFT   : in FIELD := DEFAULT_AFT;
                 EXP   : in FIELD := DEFAULT_EXP);

  procedure GET (FROM : in STRING;
                 ITEM  : out NUM;
                 LAST  : out POSITIVE);

  procedure PUT (TO : out STRING;
                 ITEM : in NUM;
                 AFT  : in FIELD := DEFAULT_AFT;
                 EXP  : in FIELD := DEFAULT_EXP);

end FIXED_IO;

pragma PAGE;
```

User's Guide
Appendix F

-- Generic Package for Input-Output of Enumeration Types

```
generic
  type ENUM is (<>);
package ENUMERATION_IO is

  DEFAULT_WIDTH   : FIELD      := 0;
  DEFAULT_SETTING : TYPE_SET := UPPER_CASE;

  procedure GET (FILE : in FILE_TYPE; ITEM : out ENUM);
  procedure GET (ITEM : out ENUM);

  procedure PUT (FILE : FILE_TYPE;
                ITEM : in ENUM;
                WIDTH : in FIELD      := DEFAULT_WIDTH;
                SET   : in TYPE_SET   := DEFAULT_SETTING);
  procedure PUT (ITEM : in ENUM;
                WIDTH : in FIELD      := DEFAULT_WIDTH;
                SET   : in TYPE_SET   := DEFAULT_SETTING);

  procedure GET (FROM : in STRING;
                ITEM : out ENUM;
                LAST : out POSITIVE);

  procedure PUT (TO : out STRING;
                ITEM : in ENUM;
                SET : in TYPE_SET := DEFAULT_SETTING);

end ENUMERATION_IO;

pragma PAGE;

-- Exceptions

STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;
LAYOUT_ERROR : exception renames IO_EXCEPTIONS.LAYOUT_ERROR;

pragma page;
private

  type FILE_TYPE is
    record
      FT : INTEGER := -1;
    end record;

end TEXT_IO;
```

F.8.2 Package IO EXCEPTIONS

The specification of the package IO_EXCEPTIONS:

```
package IO_EXCEPTIONS is  
  
  STATUS_ERROR : exception;  
  MODE_ERROR   : exception;  
  NAME_ERROR    : exception;  
  USE_ERROR     : exception;  
  DEVICE_ERROR  : exception;  
  END_ERROR     : exception;  
  DATA_ERROR   : exception;  
  LAYOUT_ERROR  : exception;  
  
end IO_EXCEPTIONS;
```


User's Guide
Appendix F

F.8.3 Package BASIC_IO

The specification of package BASIC_IO:

with IO_EXCEPTIONS;

package BASIC_IO is

 type count is range 0 .. integer'last;

 subtype positive_count is count range 1 .. count'last;

 function get_integer return string;

 -- Skips any leading blanks, line terminators or page
 -- terminators. Then reads a plus or a minus sign if
 -- present, then reads according to the syntax of an
 -- integer literal, which may be based. Stores in item
 -- a string containing an optional sign and an integer
 -- literal.

 --
 -- The exception DATA_ERROR is raised if the sequence
 -- of characters does not correspond to the syntax
 -- described above.

 --
 -- The exception END_ERROR is raised if the file terminator
 -- is read. This means that the starting sequence of an
 -- integer has not been met.

 --
 -- Note that the character terminating the operation must
 -- be available for the next get operation.

 function get_real return string;

 -- Corresponds to get_integer except that it reads according
 -- to the syntax of a real literal, which may be based.

 function get_enumeration return string;

 -- Corresponds to get_integer except that it reads according
 -- to the syntax of an identifier, where upper and lower
 -- case letters are equivalent to a character literal
 -- including the apostrophes.

User's Guide
Appendix F

```
function get_item (length : in      integer) return string;

-- Reads a string from the current line and stores it in
-- item.  If the remaining number of characters on the
-- current line is less than length then only these
-- characters are returned.  The line terminator is not
-- skipped.

procedure put_item (item : in      string);

-- If the length of the string is greater than the current
-- maximum line (linelength), the exception LAYOUT_ERROR
-- is raised.
--
-- If the string does not fit on the current line a line
-- terminator is output, then the item is output.

-- Line and page lengths - ARM 14.3.3.
--

procedure set_line_length (to      : in      count);

procedure set_page_length (to      : in      count);

function line_length return count;

function page_length return count;

-- Operations on columns, lines and pages - ARM 14.3.4.
--

procedure new_line;

procedure skip_line;

function end_of_line return boolean;

procedure new_page;

procedure skip_page;

function end_of_page return boolean;
```

User's Guide
Appendix F

```
function end_of_file return boolean;
```

```
procedure set_col (to      : in      positive_count);
```

```
procedure set_line (to      : in      positive_count);
```

```
function col return positive_count;
```

```
function line return positive_count;
```

```
function page return positive_count;
```

```
-- Character and string procedures.  
-- Corresponds to the procedures defined in ARM 14.3.6.  
--
```

```
procedure get_character (item :      out character);
```

```
procedure get_string (item :      out string);
```

```
procedure get_line (item :      out string;  
                   last :      out natural);
```

```
procedure put_character (item : in      character);
```

```
procedure put_string (item : in      string);
```

```
procedure put_line (item : in      string);
```

```
-- exceptions:
```

```
USE_ERROR      : exception renames IO_EXCEPTIONS.USE_ERROR;  
DEVICE_ERROR   : exception renames IO_EXCEPTIONS.DEVICE_ERROR;  
END_ERROR      : exception renames IO_EXCEPTIONS.END_ERROR;  
DATA_ERROR     : exception renames IO_EXCEPTIONS.DATA_ERROR;  
LAYOUT_ERROR   : exception renames IO_EXCEPTIONS.LAYOUT_ERROR;
```

```
end BASIC_IO;
```

F.8.4 Package LOW_LEVEL_IO

The specification of LOW_LEVEL_IO is:

with SYSTEM;

package LOW_LEVEL_IO is

 subtype port_address is System.Word;

 type byte is new integer;

 procedure send_control(device : in port_address;
 data : in System.Word);

 procedure send_control(device : in port_address;
 data : in byte);

 procedure receive_control(device : in port_address;
 data : out byte);

 procedure receive_control(device : in port_address;
 data : out System.Word);

 private

 pragma(inline(send_control, receive_control));

end LOW_LEVEL_IO;

User's Guide
Appendix F

F.8.5 Package TERMINAL DRIVER

The specification of package TERMINAL_DRIVER:

```
package TERMINAL_DRIVER is
  procedure put_character (ch : in character);
  procedure get_character (ch : out character);

private
  pragma interface (ASM86, put_character);
  pragma interface (ASM86, get_character);
end TERMINAL_DRIVER;
```

User's Guide
Appendix F

F.8.6 Package SEQUENTIAL_IO

```
-- Source code for SEQUENTIAL_IO

pragma PAGE;

with IO_EXCEPTIONS;

generic

    type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

    type FILE_TYPE is limited private;

    type FILE_MODE is (IN_FILE, OUT_FILE);

pragma PAGE;
-- File management

    procedure CREATE(FILE : in out FILE_TYPE;
                     MODE : in     FILE_MODE := OUT_FILE;
                     NAME : in     STRING   := "";
                     FORM : in     STRING   := "");

    procedure OPEN  (FILE : in out FILE_TYPE;
                     MODE : in     FILE_MODE;
                     NAME : in     STRING;
                     FORM : in     STRING := "");

    procedure CLOSE (FILE : in out FILE_TYPE);

    procedure DELETE(FILE : in out FILE_TYPE);

    procedure RESET (FILE : in out FILE_TYPE;
                     MODE : in     FILE_MODE);

    procedure RESET (FILE : in out FILE_TYPE);

    function MODE  (FILE : in FILE_TYPE) return FILE_MODE;

    function NAME  (FILE : in FILE_TYPE) return STRING;

    function FORM  (FILE : in FILE_TYPE) return STRING;

    function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

pragma PAGE;
-- input and output operations
```

User's Guide
Appendix F

```
procedure READ (FILE : in FILE_TYPE;  
               ITEM : out ELEMENT_TYPE);  
  
procedure WRITE (FILE : in FILE_TYPE;  
                ITEM : in ELEMENT_TYPE);  
  
function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;  
  
pragma PAGE;  
-- exceptions  
  
STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;  
MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;  
NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;  
USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;  
DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;  
END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;  
DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;  
  
pragma PAGE;  
private  
  
type FILE_TYPE is new INTEGER;  
  
end SEQUENTIAL_IO;
```

F.9 Machine Code Insertions

The reader should be familiar with the code generation strategy and the 80x86 instruction set to fully benefit from this section.

As described in chapter 13.8 of the ARM [DoD 83] it is possible to write procedures containing only code statements using the predefined package MACHINE_CODE. The package MACHINE_CODE defines the type MACHINE_INSTRUCTION which, used as a record aggregate, defines a machine code insertion. The following sections list the type MACHINE_INSTRUCTION and types on which it depends, give the restrictions, and show an example of how to use the package MACHINE_CODE.

F.9.1 Predefined Types for Machine Code Insertions

The following types are defined for use when making machine code insertions (their type declarations are given in the following pages):

```
type opcode_type
type operand_type
type register_type
type segment_register
type machine_instruction
```

The type REGISTER_TYPE defines registers and register combinations. The double register combinations (e.g. BX_SI) can be used only as address operands (BX_SI describing [BX+SI]). The registers STi describe registers on the floating stack. (ST is the top of the floating stack).

The type SEGMENT_REGISTER defines the four segment registers that can be used to overwrite default segments in an address operand.

The type MACHINE_INSTRUCTION is a discriminant record type with which every kind of instruction can be described. Symbolic names may be used in the form

name'ADDRESS

Restrictions as to symbolic names can be found in section F.9.2.

User's Guide
Appendix F

```

type opcode_type is (
    -- 8086 instructions:
    m_AAA,      m_AAD,      m_AAM,      m_AAS,
    m_ADC,      m_ADD,      m_AND,
    m_CALL,     m_CALLn,
    m_CBW,      m_CLC,      m_CLD,      m_CLI,
    m_CMC,      m_CMP,      m_CMPS,     m_CWD,
    m_DAA,      m_DAS,
    m_DEC,      m_DIV,      m_HLT,
    m_IDIV,     m_IMUL,     m_IN,      m_INC,
    m_INT,      m_INT0,     m_IRET,
    m_JA,       m_JAE,      m_JB,      m_JBE,
    m_JC,       m_JCXZ,     m_JE,      m_JG,
    m_JGE,      m_JL,       m_JLE,     m_JNA,
    m_JNAE,     m_JNB,     m_JNBE,   m_JNC,
    m_JNE,      m_JNG,     m_JNGE,   m_JNL,
    m_JNLE,     m_JNO,     m_JNP,    m_JNS,
    m_JNZ,      m_JO,      m_JP,     m_JPE,
    m_JPO,      m_JS,      m_JZ,     m_JMP,
    m_LAHF,     m_LDS,     m_LES,    m_LEA,
    m_LOCK,     m_LODS,
    m_LOOP,     m_LOOPE,   m_LOOPNE, m_LOOPNZ,
    m_LOOPZ,    m_MOV,     m_MOVS,   m_MUL,
    m_NEG,      m_NOP,     m_NOT,    m_OR,
    m_OUT,      m_POP,     m_POPF,   m_PUSH,
    m_PUSHF,
    m_RCL,      m_RCR,     m_ROL,    m_ROR,
    m_REP,      m_REPE,     m_REPNE,
    m_RET,      m_RETP,    m_RETN,   m_RETNP,
    m_SAHF,
    m_SAL,      m_SAR,     m_SHL,    m_SHR,
    m_SBB,      m_SCAS,
    m_STC,      m_STD,     m_STI,    m_STOS,
    m_SUB,      m_TEST,    m_WAIT,   m_XCHG,
    m_XLAT,     m_XOR,

```

User's Guide Appendix F

-- 8087/80187/80287 Floating Point Processor instructions

m_FABS,	m_FADD,	m_FADDD,	m_FADDP,
m_FBLD,	m_FBSTP,	m_FCHS,	m_FNCLEX,
m_FCOM,	m_FCOMD,	m_FCOMP,	m_FCOMPDP,
m_FCOMPP,	m_FDECSTP,	m_FDIV,	m_FDIVD,
m_FDIVP,	m_FDIVR,	m_FDIVRD,	m_FDIVRP,
m_FFREE,	m_FIADD,	m_FIADDD,	m_FICOM,
m_FICOMD,	m_FICOMP,	m_FICOMPDP,	m_FIDIV,
m_FIDIVD,	m_FIDIVR,	m_FIDIVRD,	
m_FILD,	m_FILDD,	m_FILDL,	m_FIMUL,
m_FIMULD,	m_FINCSTP,	m_FNINIT,	m_FIST,
m_FISTD,	m_FISTP,	m_FISTPD,	m_FISTPL,
m_FISUB,			
m_FISUBD,	m_FISUBR,	m_FISUBRD,	m_FLD,
m_FLDD,	m_FLDCW,	m_FLDENV,	m_FLDLG2,
m_FLDLN2,	m_FLDL2E,	m_FLDL2T,	m_FLDPI,
m_FLDZ,	m_FLD1,	m_FMUL,	m_FMULD,
m_FMULP,	m_FNOP,	m_FPATAN,	m_FPREM,
m_FPTAN,	m_FRNDINT,	m_FRSTOR,	m_FSAVE,
m_FSCALE,	m_FSETPM,	m_FSQRT,	
m_FST,	m_FSTD,	m_FSTCW,	
m_FSTENV,	m_FSTP,	m_FSTPD,	m_FSTSW,
m_FSTSWAX,	m_FSUB,	m_FSUBD,	m_FSUBP,
m_FSUBR,	m_FSUBRD,	m_FSUBRP,	m_FTST,
m_FWAIT,	m_FXAM,	m_FXCH,	m_FXTRACT,
m_FYL2X,	m_FYL2XP1,	m_F2XM1,	

-- 80186/80286/80386 instructions:

- Notice that some immediate versions of the 8086 instructions
- only exist on these targets (shifts, rotates, push, imul, ...)

m_BOUND,	m_CLTS,	m_ENTER,	m_INS,
m_LAR,	m_LEAVE,	m_LGDT,	m_LIDT,
m_LSL,	m_OUTS,	m_POPA,	m_PUSHA,
m_SGDT,	m_SIDT,		
m_ARPL,	m_LLDT,	m_LMSW,	m_LTR,
m_SLDT,	m_SMSW,	m_STR,	m_VERR,
m_VERW,			

User's Guide
Appendix F

-- the 80386 specific instructions:

m_SETA,	m_SETAE,	m_SETB,	m_SETBE,
m_SETC,	m_SETE,	m_SETG,	m_SETGE,
m_SETL,	m_SETLE,	m_SETNA,	m_SETNAE,
m_SETNB,	m_SETNBE,	m_SETNC,	m_SETNE,
m_SETNG,	m_SETNGE,	m_SETNL,	m_SETNLE,
m_SETNO,	m_SETNP,	m_SETNS,	m_SETNZ,
m_SETO,	m_SETP,	m_SETPE,	m_SETPO,
m_SETS,	m_SETZ,		
m_BSF,	m_BSR,		
m_BT,	m_BTC,	m_BTR,	m_BTS,
m_LFS,	m_LGS,	m_LSS,	
m_MOVZX,	m_MOVSX,		
m_MOVCR,	m_MOVDB,	m_MOVTR,	
m_SHLD,	m_SHRD,		

-- the 80387 specific instructions

m_FUCOM,	m_FUCOMP,	m_FUCOMPP	
m_FPREM1,	m_FSIN,	m_FCOS,	m_FSINCOS

-- byte/word/dword variants (to be used, when not deductible from
-- context

m_ADDCB,	m_ADCW,	m_ADCD,
m_ADDB,	m_ADDW,	m_ADDD,
m_ANDB,	m_ANDW,	m_ANDD,
	m_BTW,	m_BTD,
	m_BTCW,	m_BTCD,
	m_BTRW,	m_BTRD,
	m_BTSW,	m_BTSD,
	m_CBWW,	m_CWDE,
	m_CWDW,	m_CDQ,
m_CMPB,	m_CMPW,	m_CMPD,
m_CMPSB,	m_CMPSW,	m_CMPSD,
m_DECB,	m_DECW,	m_DECD,
m_DIVB,	m_DIVW,	m_DIVD,
m_IDIVB,	m_IDIVW,	m_IDIVD,
m_IMULB,	m_IMULW,	m_IMULD,
m_INCB,	m_INCW,	m_INCD,
m_INSB,	m_INSW,	m_INSD,
m_LODSB,	m_LODSW,	m_LOSD,
m_MOVB,	m_MOVW,	m_MOVD,
m_MOVSB,	m_MOVSW,	m_MOVSD,
m_MOVSXB,	m_MOVSXW,	
m_MOVZXB,	m_MOVZXW,	
m_MULB,	m_MULW,	m_MULD,
m_NEGB,	m_NEGW,	m_NEGD,
m_NOTB,	m_NOTW,	m_NOTD,
m_ORB,	m_ORW,	m_ORD,
m_OUTSB,	m_OUTSW,	m_OUTSD,
	m_POPW,	m_POPD,
	m_PUSHW,	m_PUSD,

User's Guide
Appendix F

m_RCLB,	m_RCLW,	m_RCLD,
m_RCRB,	m_RCRW,	m_RCRD,
m_ROLB,	m_ROLW,	m_ROLD,
m_RORB,	m_RORW,	m_RORD,
m_SALB,	m_SALW,	m_SALD,
m_SARB,	m_SARW,	m_SARD,
m_SHLB,	m_SHLW,	m_SHLDW,
m_SHRB,	m_SHRW,	m_SHRDW,
m_SBBB,	m_SBBW,	m_SBBD,
m_SCASB,	m_SCASW,	m_SCASD,
m_STOSB,	m_STOSW,	m_STOSD,
m_SUBB,	m_SUBW,	m_SUBD,
m_TESTB,	m_TESTW,	m_TESTD,
m_XORB,	m_XORW,	m_XORD,
m_DATAB,	m_DATAW,	m_DATAD

-- Special 'instructions'

m_label, m_reset);

pragma page;

User's Guide
Appendix F

```

type operand_type is (none,                -- no operands

    immediate,                             -- 1 immediate operand
    register,                              -- 1 register operand
    address,                               -- 1 address operand
    system_address,                        -- 1 'address operand
    name,                                  -- CALL name
    register_immediate,                   -- 2 operands: dest is
                                         -- register, source is
                                         -- immediate
    register_register,                    -- 2 register operands
    register_address,                     -- 2 operands: dest is
                                         -- register, source is
                                         -- address
    address_register,                     -- 2 operands: dest is
                                         -- address, source is
                                         -- register
    register_system_address,              -- 2 operands: dest is
                                         -- register, source is
                                         -- 'address
    system_address_register,              -- 2 operands: dest is
                                         -- 'address, source is
                                         -- register
    address_immediate,                    -- 2 operands: dest is
                                         -- 'address, source is
                                         -- immediate
    system_address_immediate,             -- 2 operands: dest is
                                         -- 'address, source is
                                         -- immediate
    immediate_register,                   -- only allowed for
                                         -- OUT
                                         -- port is immediate
                                         -- source is register
    immediate_immediate,                  -- only allowed for
                                         -- ENTER
    register_register_immediate,          -- allowed for IMULimm,
                                         -- SHRDImm, and SHLDImm
    register_address_immediate            -- allowed for IMULImm
    register_system_address_immediate     -- allowed for IMULImm
    address_register_immediate            -- allowed for SHRDImm,
                                         -- SHLDImm
    system_address_register_immediate     -- allowed for SHRDImm,
                                         -- SHLDImm
);

```

```

type register_type is (AX, CX, DX, BX,    -- word registers
    SP, BP, SI, DI,                      --
                                         --
    AL, CL, DL, BL,                      -- byte registers
    AH, CH, DH, BH,                      --
                                         --
    EAX, ECX, EDX, EBX -- dword registers
);

```

User's Guide
Appendix F

ESP, EBP, ESI, EDI

ES, CS, SS, DS, -- selector registers
FS, GS

BX_SI, BX_DI, -- 8086/80186/80286
BP_SI, BP_DI, -- combinations

ST, ST1, ST2, ST3, -- floating
 -- stack
 -- registers

ST4, ST5, ST6, ST7,

nil);

type segment_register is (ES, CS, SS, DS, FS, GS, nil);
-- segment registers
-- FS and GS are only allowed in 80386 targets

subtype machine_string is string (1..100);

pragma page;

User's Guide
Appendix F

```
type machine_instruction (operand_kind : operand_type is
  record
    opcode : opcode_type;

    case operand_kind is
      when immediate =>
        immediate      : integer;

      when register =>
        r_register      : register_type;

      when address =>
        a_segment       : register_type;
        a_address_reg   : register_type;
        a_offset        : integer;

      when system_address =>
        sa_address      : system.address;

      when name =>
        n_string        : machine_string;

      when register_immediate =>
        r_i_register    : register_type;
        r_i_immediate   : integer;

      when register_register =>
        r_r_register_to : register_type;
        r_r_register_from : register_type;

      when register_address =>
        r_a_register_to : register_type;
        r_a_segment     : segment_register;
        r_a_address_reg : register_type;
        r_a_offset      : integer;

      when address_register =>
        a_r_segment     : segment_register;
        a_r_address_reg : register_type;
        a_r_offset      : integer;
        a_r_register_from : register_type;

      when register_system_address =>
        r_sa_register_to : register_type;
        r_sa_address     : system.address;

      when system_address_register =>
        sa_r_address    : system.address;
        sa_r_reg_from   : register_type;
```

User's Guide
Appendix F

```
when address_immediate =>
  a_i_segment      : segment_register;
  a_i_address_reg   : register_type;
  a_i_offset        : integer;
  a_i_immediate     : integer;

when system_address_immediate =>
  sa_i_address      : system.address;
  sa_i_immediate     : integer;

when immediate_register =>
  i_r_register      : integer;
  i_r_register      : register_type;

when immediate_immediate =>
  i_i_immediate1     : integer;
  i_i_immediate2     : integer;

when register_register_immediate =>
  r_r_i_register1    : register_type;
  r_r_i_register2    : register_type;
  r_r_i_immediate2    : integer;

when register_address_immediate =>
  r_a_i_register     : register_type;
  r_a_i_segment      : register_type;
  r_a_i_address_reg   : register_type;
  r_a_i_offset        : integer;
  r_a_i_immediate     : integer;

when register_system_address_immediate =>
  r_sa_i_register     : register_type;
  addr10              : system.address;
  r_sa_i_immediate     : integer;

when address_register_immediate =>
  a_r_i_register      : register_type;
  a_r_i_segment      : register_type;
  a_r_i_address_reg   : register_type;
  a_r_i_offset        : integer;
  a_r_i_immediate     : integer;

when system_address_register_immediate =>
  sa_r_i_address      : system.address;
  sa_r_i_register     : register_type;
  sa_r_i_immediate     : integer;

when others =>
  null;
end case;
end record;
```


User's Guide
Appendix F

F.9.2 Restrictions

Only procedures, and not functions, may contain machine code insertions. Also procedures that use machine code insertions must be inline.

Symbolic names in the form x'ADDRESS can only be used in the following cases:

- 1) x is an object of scalar type or access type declared as an object, a formal parameter, or by static renaming.
- 2) x is an array with static constraints declared as an object (not as a formal parameter or by renaming).
- 3) x is a record declared as an object (not a formal parameter or by renaming).

All opcodes defined by the type OPCODE_type except the m_CALL can be used.

Two opcodes to handle labels have been defined:

- m_label: defines a label. The label number must be in the range $1 \leq x \leq 25$ and is put in the offset field in the first operand of the MACHINE_INSTRUCTION.
- m_reset: used to enable use of more than 25 labels. The label number after a m_RESET must be in the range $1 \leq x \leq 25$. To avoid errors you must make sure that all used labels have been defined before a reset, since the reset operation clears all used labels.

All floating instructions have at most one operand which can be any of the following:

- a memory address
- a register or an immediate value
- an entry in the floating stack

User's Guide
Appendix F

F.9.3 Examples

The following section contains examples of how to use the machine code insertions and lists the generated code.

F.9.3.1 Example Using Labels

The following assembler code can be described by machine code insertions as shown:

```
MOV AX,7
MOV CX,4
CMP AX,CX
JG 1
JE 2
MOV CX,AX
1: ADD AX,CX
2: MOV SS: [BP+DI], AX
```

with MACHINE_CODE; use MACHINE_CODE;
package example_MC is

```
    procedure test_labels;
    pragma inline (test_labels);
```

end example_MC;

package body example_MC is

procedure test_labels is

begin

```
    MACHINE_INSTRUCTION'(register_immediate, m_MOV, AX, 7);
    MACHINE_INSTRUCTION'(register_immediate, m_MOV, CX, 4);
    MACHINE_INSTRUCTION'(register_register, m_CMP, AX, CX);
    MACHINE_INSTRUCTION'(immediate, m_JG, 1);
    MACHINE_INSTRUCTION'(immediate, m_JE, 2);
    MACHINE_INSTRUCTION'(register_register, m_MOV, CX, AX);
    MACHINE_INSTRUCTION'(immediate, m_label, 1);
    MACHINE_INSTRUCTION'(register_register, m_ADD, AX, CX);
    MACHINE_INSTRUCTION'(immediate, m_label, 2);
    MACHINE_INSTRUCTION'(address_register, m_MOV, SS, BP_DI, 0, AX);
```

end test_labels;

end example_MC;

User's Guide
Appendix F

F.10 Package Tasktypes

The TaskTypes packages defines the TaskControlBlock type. This data structure could be useful in debugging a tasking program. The following package Tasktypes is for the 80x86 Real Address Mode and 80286 Protected Mode systems:

with System;

package TaskTypes is

```
    subtype Offset      is System.Word;
    subtype BlockId     is System.Word;

    type TaskEntry      is new System.Word;
    type EntryIndex     is new System.Word;
    type AlternativeId  is new System.Word;
    type Ticks          is new System.LongWord;

    type TaskState      is (Initial,
                           Engaged,
                           Running,
                           Delayed,
                           EntryCallingTimed,
                           EntryCallingUnconditional,
                           SelectingTimed,
                           SelectingTerminable,
                           Accepting,
                           Synchronizing,
                           Completed,
                           Terminated);
```

```
    type TaskTypeDescriptor is
        record
            priority          : System.Priority;
            entry_count       : System.Word;
            block_id          : BlockId;
            first_own_address : System.Address;
            module_number     : System.Word;
            entry_number      : System.Word;
            code_address       : System.Address;
            stack_size        : System.LongWord;
            stack_segment_size : System.Word;
        end record;
```

```
    type NPXSaveArea is array(1..48) of System.Word;
```

```
pragma page;
    type TaskControlBlock is
        record
            sem : System.Semaphore;
```

User's Guide
Appendix F

```
-- Delay queue handling

    dnext          : System.TaskValue ;
    dprev          : System.TaskValue ;
    ddelay         : Ticks ;

-- Saved registers

    SS             : System.Word ;
    SP             : System.Word ;

-- Ready queue handling

    next           : System.TaskValue ;

-- Semaphore handling

    semnext        : System.TaskValue ;

-- Priority fields

    priority        : System.Priority;
    saved_priority  : System.Priority;

    time_slice      : Ticks;

    stack_start     : Offset;
    stack_end       : Offset;

-- State fields

    state           : TaskState;
    is_abnormal     : Boolean;
    is_activated    : Boolean;
    failure         : Boolean;

-- Activation handling fields

    activator       : System.TaskValue;
    act_chain       : System.TaskValue;
    next_chain      : System.TaskValue;
    no_not_act      : System.Word;
    act_block       : BlockId;

-- Accept queue fields

    partner         : System.TaskValue;
    next_partner    : System.TaskValue;
```

User's Guide
Appendix F

```
-- Entry queue fields

    next_caller      : System.TaskValue;

-- Rendezvous fields

    called_task      : System.TaskValue;
    task_entry       : TaskEntry;
    entry_index      : EntryIndex;
    entry_assoc      : System.Address;
    call_params      : System.Address;
    alt_id           : AlternativeId;
    excp_id          : System.ExceptionId;

-- Dependency fields

    parent_task      : System.TaskValue;
    parent_block     : BlockId;
    child_task       : System.TaskValue;
    next_child       : System.TaskValue;
    first_child      : System.TaskValue;
    prev_child       : System.TaskValue;
    child_act        : System.Word;
    block_act        : System.Word;
    terminated_task  : System.TaskValue;

-- Abortion handling fields

    busy             : System.Word;

-- Auxiliary fields

    ttd              : TaskTypeDescriptor;
    segment_size     : System.Word;

-- Run-Time System fields

    ACF              : Offset;
    collection       : System.Address;

-- NPX save area
    NPXSave          : NPXSaveArea;
end record;

end TaskTypes;
```

User's Guide
Appendix F

The following package Tasktypes is for the 80386 Protected Mode system:

with System;

package TaskTypes is

```
    subtype Offset      is System.UnsignedDWord;
    subtype BlockId     is System.UnsignedDWord;

    type TaskEntry      is new System.UnsignedDWord;
    type EntryIndex     is new System.UnsignedDWord;
    type AlternativeId  is new System.UnsignedDWord;
    type Ticks          is new System.UnsignedDWord;
    type Bool           is new Boolean;
    for Bool'size       use 8;
    type UIntg          is new System.UnsignedDWord;

    type TaskState      is (Initial,
                           Engaged,
                           Running,
                           Delayed,
                           EntryCallingTimed,
                           EntryCallingUnconditional,
                           SelectingTimed,
                           SelectingTerminable,
                           Accepting,
                           Synchronizing,
                           Completed,
                           Terminated);
```

```
type TaskTypeDescriptor is
    record
        priority      : System.Priority;
        entry_count   : UIntg;
        block_id      : BlockId;
        first_own_address : System.Address;
        module_number : UIntg;
        entry_number   : UIntg;
        code_address   : System.Address;
        stack_size     : System.QWord;
        stack_segment_size: UIntg;
    end record;
```

```
type NPXSaveArea is array(1..48) of System.Word;
```

```
pragma page;
```

```
type TaskControlBlock is
```

```
    record
        sem          : System.Semaphore;
```

User's Guide
Appendix F

-- Delay queue handling

dnext : System.TaskValue ;
dprev : System.TaskValue ;
ddelay : Ticks ;

-- Saved registers

SS : System.Word ;
SP : Offset ;

-- Ready queue handling

next : System.TaskValue ;

-- Semaphore handling

semnext : System.TaskValue ;

-- Priority fields

priority : System.Priority;
saved_priority : System.Priority;

time_slice : Ticks;

stack_start : Offset;
stack_end : Offset;

-- State fields

state : TaskState;
is_abnormal : Bool;
is_activated : Bool;
failure : Bool;

-- Activation handling fields

activator : System.TaskValue;
act_chain : System.TaskValue;
next_chain : System.TaskValue;
no_not_act : System.Word;
act_block : BlockId;

-- Accept queue fields

partner : System.TaskValue;
next_partner : System.TaskValue;

User's Guide
Appendix F

```
-- Entry queue fields

    next_caller      : System.TaskValue;

-- Rendezvous fields

    called_task      : System.TaskValue;
    task_entry       : TaskEntry;
    entry_index      : EntryIndex;
    entry_assoc      : System.Address;
    call_params      : System.Address;
    alt_id           : AlternativeId;
    excp_id          : System.ExceptionId;

-- Dependency fields

    parent_task      : System.TaskValue;
    parent_block     : BlockId;
    child_task       : System.TaskValue;
    next_child       : System.TaskValue;
    first_child      : System.TaskValue;
    prev_child       : System.TaskValue;
    child_act        : System.Word;
    block_act        : System.Word;
    terminated_task   : System.TaskValue;

-- Abortion handling fields

    busy             : System.Word;

-- Auxiliary fields

    ttd              : TaskTypeDescriptor;
    segment_size     : System.Word;

-- Run-Time System fields

    ACF              : Offset;
    collection       : System.Address;

-- NPX save area
    NPXSave          : NPXSaveArea;
end record;

end TaskTypes;
```


APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

\$ACC_SIZE	32
An integer literal whose value is the number of bits sufficient to hold any value of an access type.	
\$BIG_ID1	1..125 => 'A', 126 => '1'
Identifier the size of the maximum input line length with varying last character.	
\$BIG_ID2	1..125 => 'A', 126 => '2'
Identifier the size of the maximum input line length with varying last character.	
\$BIG_ID3	1..63 => 'A', 64 => '3', 65..126 => 'A'
Identifier the size of the maximum input line length with varying middle character.	
\$BIG_ID4	1..63 => 'A', 64 => '4', 65..126 => 'A'
Identifier the size of the maximum input line length with varying middle character.	
\$BIG_INT_LIT	0..123 => 0, 124..126 => 298
An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	
\$BIG_REAL_LIT	0..120 => 0, 121..126 => 69.0E1
A universal real literal of value 690.0 with enough leading zeroes to be the size of the	

maximum line length.

\$BIG_STRING1	0..63 => 'A'
A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	
\$BIG_STRING2	0..62 => 'A', 63 => '1'
A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	
\$BLANKS	0..106 => ' '
A sequence of blanks twenty characters less than the size of the maximum line length.	
\$COUNT_LAST	32767
A universal integer literal whose value is TEXT_IO.COUNT'LAST.	
\$DEFAULT_MEM_SIZE	1_048_576
An integer literal whose value is SYSTEM.MEMORY_SIZE.	
\$DEFAULT_STOR_UNIT	16
An integer literal whose value is SYSTEM.STORAGE_UNIT.	
\$DEFAULT_SYS_NAME	IAPX186
The value of the constant SYSTEM.SYSTEM_NAME.	
\$DELTA_DOC	2#1.0#E-31
A real literal whose value is SYSTEM.FINE_DELTA.	
\$FIELD_LAST	35
A universal integer literal whose value is TEXT_IO.FIELD'LAST.	
\$FIXED_NAME	NO_SUCH_TYPE
The name of a predefined fixed-point type other than DURATION.	
\$FLOAT_NAME	NO_SUCH_TYPE
The name of a predefined floating-point type other than	

FLOAT, SHORT_FLOAT, or LONG_FLOAT.	
\$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100000.0
\$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200000.0
\$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	31
\$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	ILLEGAL!@#\$%^
\$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	ILLEGAL&() +-
\$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
\$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
\$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32_768
\$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100000.0
\$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-200000.0
\$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0

\$MANTISSA_DOC	31
An integer literal whose value is SYSTEM.MAX_MANTISSA.	
\$MAX_DIGITS	15
Maximum digits supported for floating-point types.	
\$MAX_IN_LEN	126
Maximum input line length permitted by the implementation.	
\$MAX_INT	2147483647
A universal integer literal whose value is SYSTEM.MAX_INT.	
\$MAX_INT_PLUS_1	2_147_483_648
A universal integer literal whose value is SYSTEM.MAX_INT+1.	
\$MAX_LEN_INT_BASED_LITERAL	1..2 => '2:', 3..123 => '0', 124..126 => '11:'
A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	
\$MAX_LEN_REAL_BASED_LITERAL	1..3 => '16:', 4..122 => '0', 123..126 => 'F.E:'
A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	
\$MAX_STRING_LITERAL	1 => '"', 2..125 => 'A', 126 => '"""
A string literal of size MAX_IN_LEN, including the quote characters.	
\$MIN_INT	-2147483648
A universal integer literal whose value is SYSTEM.MIN_INT.	
\$MIN_TASK_SIZE	16
An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	
\$NAME	NO_SUCH_TYPE

A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.

\$NAME_LIST

IAPX186

A list of enumeration literals in the type SYSTEM.NAME, separated by commas.

\$NEG_BASED_INT

16#FFFFFFFF#

A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.

\$NEW_MEM_SIZE

1_048_576

An integer literal whose value is a permitted argument for pragma memory_size, other than \$DEFAULT_MEM_SIZE. If there is no other value, then use \$DEFAULT_MEM_SIZE.

\$NEW_STOR_UNIT

16

An integer literal whose value is a permitted argument for pragma storage_unit, other than \$DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.

\$NEW_SYS_NAME

IAPX186

A value of the type SYSTEM.NAME, other than \$DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.

\$TASK_SIZE

16

An integer literal whose value is the number of bits required to hold a task object which has a single entry with one inout parameter.

\$TICK

0.000_000_125

A real literal whose value is SYSTEM.TICK.

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 43 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G

This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E

This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

BC3009B

This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D

This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]

These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110

These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C

These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B

This test gives a SMALL representation clause for a derived fixed-point

type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B

This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]

These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A

This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B

These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D

This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I

This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C

This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A

This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B

This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C

This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E
COMPILER OPTIONS AS SUPPLIED BY

DDC-I, Inc

Compiler: DACS-80186 Version 4.3

ACVC Version: 1.10

OPTION	EFFECT
/CHECK /NOCHECK	Generates run-time constraint checks.
/CONFIGURATION_FILE	Specifies the file used by the compiler.
/DEBUG /NODEBUG	Include the symbolic debugging in the program library.
/EXCEPTION_TABLES /NOEXCEPTION_TABLES	Include/exclude exception handler tables from the generated code.
/LIBRARY	Specify program library used.
/LIST /NOLIST	Write a source listing on the list file.
/OPTIMIZE /NOOPTIMIZE	Specify compiler optimization
/PROGRESS /NOPROGRESS	Display compiler progress
/SAVE_SOURCE /NOSAVE_SOURCE	Copies source to program library
/XREF /NOXREF	Creates a cross reference listing